Work-In-Progress: Real-Time RPC for Hybrid Dual-OS System

Pan Dong[12] Zhe Jiang[1] Alan Burns[1] Yan Ding[2] Jun Ma[2]
[1] Computer Science Department, University of York, YO10 5GH, UK
[2]School of Computer, National University of Defense Technology, Changsha, Hunan Province, P.R.China

Abstract—**For the power and space sensitive systems such as automotive/avionic computers, an important trend is isolating and integrating multiple Operating Systems (OSs) in one physical platform, which is named as hybrid multi-OS system. Generally, in a commonly used hybrid dual-OS system, a RTOS (real-time operating system) and a GPOS (general-purpose operating system) are integrated. Cooperation (among the OSs) is a vital feature of a hybrid system to obtain the necessary capabilities, and inter-OS communication is the key. However, it is difficult to satisfy the real-time metrics of inter-OS communication required by the RTOS, due to the uncertainty in communication maintenance and the time-sharing policy of the GPOS. This paper aims to build a time predictable and secure RPC mechanism (i.e., the primary and critical communication unit in a hybrid multi-OS system). Afterwards, a real-time RPC scheme (termed RTRG-RPC) is proposed, which is applied to a ready-built TrustZone-based hybrid dual-OS system (i.e., TZDKS). RTRG-RPC achieves accurate time control through three mechanisms: SGI message transforming, interrupt handler RPC servicing, and priority-swapping. Evaluations show that RTRG-RPC can achieve real-time predictability and can also reduce priority inversion.**

I. Introduction

A major trend in automotive/avionic system is the consolidation of multiple domains on single powerful SoCs [1], in order to optimize cost, space, weight, heat generation, and power consumption. The up-to-date ARINC 653 [9] specification requires integrating flight control systems, environment control systems, and amusement systems into a virtualized platform on modern aircraft. Meanwhile, AUTOSAR 4.0+ [1] proposes ECUs (Electronic Control Units) consolidation in a car based on virtualization. Even more, the consolidation of ECUs, ADAS (Advanced Driver-Assistance System) and IVIS (In-Vehicle Information System) subsystems will be the final target [1]. Therefore, the final platform is a hybrid system with different characteristics, and is termed as hybrid multi-OS system in some research [11]. A popular candidate for consolidation is isolating the sub-system into the different run-time environments (e.g., virtualization [6] – running each sub-system in independent virtual machines). However, this method significantly conflicts with the requirements on resource efficiency and predictability, due to the introduction of complicated resource management and complex access paths [5], [7]. The other method is building the sub-systems in the isolated environment provided as the extension by the SoC hardware, such as ARM TrustZone. For example, TZDKS [2] and LTZVisor [8] proposed multi-OS architectures upon TrustZone, which achieve better system performance.

The simplest form of a hybrid multi-OS system is the composition of a real-time OS (RTOS) and a general-purpose OS (GPOS), i.e., a hybrid dual-OS system. However, the consolidation is not merely a simple composition of OSs. Because both OSs will benefit from the inter-operations in terms of functions and performance, we can get a new system with the result one plus one is greater than two. Take the automotive as an example, with the assistance of inter-operation, the ECU cluster in the RTOS can acquire abundant functionalities (such as fault logs, cloud-side AI decision, etc.) from the IVIS in the GPOS [10]. In a hybrid multi-OS system, communication is the foundation of inter-operation, and RPC has become a fundamental mechanism [10]. Security and efficiency are the two most important metrics for communication [2]. We note that the time predictability for communication is necessary, because the scheduler should be able to predict the duration time of the communication on the RTOS part. As far as we know, there is very few literature addressing such problems,which may be a significant obstacle to promote the development of the hybrid multi-OS system.

The contributions of this paper are summarized as:

- A RPC model for the time and security analysis in the hybrid multi-OS system.
- A Real-Time RTOS-GPOS RPC protocol (i.e., RTRG-RPC) on TZDKS [2], with three main mechanisms: SGI (Software Generate Interrupt) messages transforming, interrupt handler RPC serving, and priority-swapping.
- Performance evaluation, showing real-time predictability and reduced priority inversion by RTRG-RPC.

II. Related Work

A. Inter-OS Communication

Current inter-OS communication mechanisms are mostly designed for virtualization systems. The default method is to route messages via the standard network interface. This offers the highest amount of isolation, yet provides the lowest performance. Many improvements simplify the under protocol stack, and use shared-memory to increase the performance [4]. Examples include XenLoop, MemPipe [13], etc. Another effort is to design straightforward RPC with direct hardware assistants. XENRPC [3] is a product of such an idea. In some dual-OS systems, the characteristics of a special platform are leveraged to build efficient communication. As shown in the design of SafeG [10], efficient dual-OS communication protocols are proposed by using SGI as the event path and using shared memory as the data path. All the above efforts are focused on optimization for high performance, and have no consideration for the predictability.

B. *Review of TZDKS*

TZDKS [2] is the product of the idea that combines strong points of dual-kernel [2] and virtualization by utilizing TrustZone technology [12]. Firstly, TZDKS adopts a dual-kernel structure to support applications of different time-sensitivities. All tasks in RTOS can be controlled in a time-deterministic way, because (i) the RTOS kernel is independently driven by a high-resolution timer, (ii)RTOS supports pre-empted fix-priority scheduling and (iii) GPOS executes integrally as a scheduling unit in RTOS. Meanwhile, idle time in RTOS will be fully utilized by GPOS to execute non-real-time applications, and the throughput or interactive strategies still work in GPOS. Secondly, TZDKS utilizes TrustZone to get performance trade-off. TrustZone extension provides isolation while ensuring efficient access to resources. As shown in Fig 1, two software stacks locate in the two worlds of TrustZone-enabled environment. The secure world stack is composed by the monitor module and RTOS, providing a real-time environment for the development of real-time applications. Meantime, the normal world stack running GPOS provides sufficient resources for execution of user interfaces, internet-based applications and services.

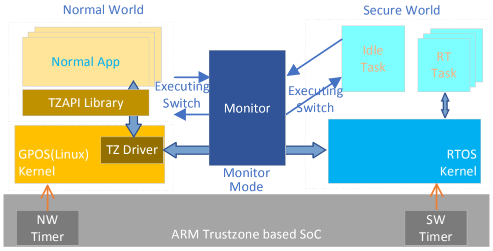

Fig. 1. TZDKS Architecture

III. PROBLEM STATEMENT

There are two different types of RPC, which is classified by the direction of communication. We only discuss the RPC from the RTOS task to an entity in GPOS, and name it RG-RPC, because in most of the cases the RG-RPC requires higher demand for time predictability [2].

An abstracted process of RG-RPC is shown in Fig 2. When a task τ_i triggers a RPC request, it will sleep until the RPC answer received, or timeout. The RG-RPG process constitutes five essential parts (sequence may be different), which are:

1) Issuing RPC request. τ_i triggers a RG-RPC, then sleeps.
2) Scheduling&switching of RTOS. Before GPOS executes.
3) GPOS executing. The RPC service will run in this period.
4) Another round of scheduling&switching by RTOS kernel. τ_i should be awoken and fed into the ready queue.
5) Return of RPC. τ_i is switched on with the RPC return.

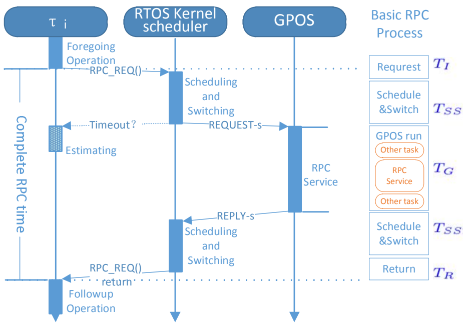

Fig. 2. Basic Process of RG-RPC on Uni-processor

We define them as T_I, T_{SS1}, T_G, T_{SS2}, T_R respectively in a sequence block as shown in the right of Fig 2.

As communication break the isolation, two aspects should be taken into account in its design: efficiency and security.

A. Efficiency. The communication has to meet the time requirement so a request can be served and returned in time. Therefore, we add some real-time constraints for requesting messages (from RTOS) and its feedback. It is difficult to satisfy the real-time constraints for communication (between RTOS and GPOS). Firstly, GPOS runs with a lower priority in a consolidation system, and it gets the CPU resources relying on RTOS (decided by T_{SS1}). Secondly, in GPOS, the RPC service time is not deterministic (T_G part). Thirdly, without preemption mechanisms, a higher priority RPC may be blocked by a lower priority one (probably in T_{SS1} or T_{SS2}).

B. Security. In practice, this property contains both safety and security. For safety, each communication must not lead to or propagate hazards and faults. For security, a malicious task can not threat other OSs or get private information through a deliberate message. In the TZDKS, the communication may break such protection and provide a window to GPOS, by which some male-wares can even intrude RTOS. Some examples [12]show how to leverage the communication and vulnerabilities to attack the TrustZone protection. Besides, over-much communication requests may trigger DoS (Denial of Service) of related OSs.

IV. DESIGN OF RTRG-RPC

A. *Design Philosophy*

The problem to minimize the T_{RG-RPC} can be transferred to the minimizing of preemptions owned by lower priority tasks (including preemptions in GPOS), although the task with higher priority is still permitted to preempt. Therefore, a new scheduling policy is required to switch GPOS on/off in time other than the idle-scheduling strategy, and three problems should be considered. 1) GPOS (normally is Linux which has no real-time scheduler) should switch on the RPC service code in the first time; 2) GPOS should efficiently inform RTOS to switch on when finishing the RPC service; 3) server side in GPOS should also support RPC priority. Moreover, as all know, page fault can affect time predictability seriously, so it should be avoided in the RPC service running.

B. Efficiency Design of RTRG-RPC

In this sub-section, specific TrustZone mechanisms are leveraged in the design of RTRG-RPC in TZDKS. However, the basic principle can be used in other consolidation systems.

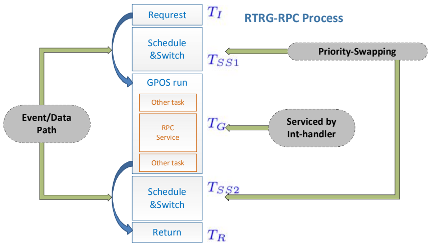

Fig. 3. Main Mechanisms for RTRG-RPC

1) Enhanced Idle-Scheduling Policy: We design an enhanced idle-scheduling mechanism for TZDKS, termed as *Priority Swapping*. It contains:

- Idle-scheduling plus τ_G. We hence add another real-time task τ_G also serving as a container of GPOS, but with a variable priority. τ_G owns a very low original priority, which will be changed temporarily when RG-RPC occurs.
- Priority-Swapping. When a regular task τ_i has triggered a RG-RPC call, it turns to sleep and exchange its priority with τ_G. Then τ_G is scheduled with the priority originally belong to τ_i. Until RPC has been served by GPOS, RTOS resets both priorities to original values, and awakes τ_i.
- Timeout-exit strategy for τ_G. A timeout value can be setup for τ_i initially. If there is no RPC response yet after a specific duration, τ_G will be suspended for timeout. Meanwhile, the priorities of τ_G and τ_i will be restored. This mechanism is proposed for the cause of the relatively lower reliability of GPOS.

2) RTRG-RPC Event Path: The latency is influenced by event path methods, especially in the GPOS part of the RPC link. The most efficient event approach is interrupt. Considering RPC call being sent to another OS but maybe on the same processor, software interrupt is chosen as the event method — currently supported by almost all hardware systems. Thus when τ_G switches the GPOS on, the SGI interrupt handler will be invoked immediately. We envelop the RG-RPC service in the SGI handler function. So the time that RPC service waits to run can be minimized. At the end of RG-RPC service, a SMC call will be triggered by the service in GPOS.

3) RTRG-RPC Data Path: RTRG-RPC uses the shared memory as the data channel, realized as a request pool and an answer pool. The pool-head is used for maintenance. The index of the head member links to the slot number in the pool, and the value of a head member remarks the priority of the RPC trigger task. We design a simple method to set up the priority of RT-RPC. The consumer just finds the minimum $prio[i]$ (the highest priority), and fetches related message in the slot. Thus, the producer has nothing to do about the priority.

4) RTRG-RPC Service implementation in GPOS: We employ the interrupt handler in the GPOS kernel to serve RG-RPC. That will avoid most indeterminate factors such as page fault and scheduling delay. Considering the system efficiency affected by long hardirq critical region, we can place the RPC service into a high priority softirq (with the hypothesis that there is a limit number of hardirq per time unit). Although GPOS is a non-real-time system designed for the maximum throughput, we are still able to make the service time determinable by increasing the priority of the interrupt related to RG-RPC, and by simplifying the procedure of RPC service into a kernel module. GICv3 hardware guarantees that the unmasked interrupt with the highest priority will be firstly sent to the CPU core in bounded time. Therefore, the GPOS service part can obtain time determinacy through the elaborate setting of interrupt priority.

C. Security Design of RTRG-RPC

Basically, three types of threat are considered. The first is the safety threat. The communication should have no side-effect of running, switch and restoring of any OS. Because RTRG-RPC only provides a transport layer for communications, the design should cope with three cases: RPC no-return, RPC wrong return, and wrong order of multiple RPCs. Our enhanced idle-scheduling policy includes the time-out exit mechanism, which will cope with no-return problem. The RPC wrong return can not be detected by this layer, so we leave it to be solved by the protocols or ways on the upper soft layer. The task ID attached in each RPC can help to solve wrong order problem, and we limit that a task can not issue the next RPC request before the finish of the current one. The second is the malicious code threat in the message, our design should prevent the executing of code in the buffer memory used by the communication. Our design only considers the data path for RTRG-RPC, because the event path can not carry any executed code. We leverage the DEP (Data Execution Protect) mechanism provided by hardware to forbid the code executing. DEP prevents code execution from data pages in the default heap, memory stacks, and memory pools. The third is the DoS attack threat. Any participant should not damage the availability of the other side OS. We set up a counter in the RPC service handler of RTOS to test the frequency of calling from GPOS. If the calling frequency exceeds a predefined threshold, RTOS will deprive some execution ticks from GPOS, so that the DoS threat can be eliminated.

V. Evaluation

We implemented TZDKS with RTRG-RPC supporting on an ARMv8 Foundation Fast-Model Platform.

A. Latency Predictability and Distribution

Two experiments are designed to verify the predictability of RTRG-RPC latency. There are three periodic tasks τ, τ_1, τ_2 in RTOS and only τ requires RPC (Note that if there are more RPC caller tasks in the RTOS, the requests will be scheduled by the RTOS kernel, and we only want to demonstrate that the predictability of RPC is almost unaffected by the low priority tasks). The total load of RTOS is less than 69%, so that RTOS is schedulable under FPS policy. In the first experiment, we give the highest priority to τ,so no preemption takes place in the process of RPC. We repeat RPC triggered by τ for 450 thousand times, meantime we use UnixBench as a payload in GPOS to get 100% CPU usage rate, and record the latency. Figure 4 summarizes the results.

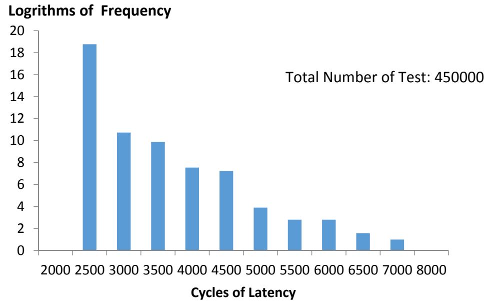

Fig. 4. Latency Distribution of RTRG-RPC

Because the data concentrates extremely (more than 99.3% calls complete in 2500 cycles), we use the logarithms of occurrence times as the Y-axis scale. As shown in Fig4, all RPCs complete in 8000 cycles, and only 3 calls exceed 6000 cycles. This result shows the predictability of RTRG-RPC.

In the second experiment, we consider the preemption by tasks with higher priority in RTOS. We assign the lowest priority to τ, and test the latency of RPC calls triggered by τ, — this is the only difference from the first experiment. In table I, we give the comparison of maximum (Max) latency, minimum (Min) latency, average latency, and the mean-square error (MSE) of latency for RTRG-RPC in two cases. We can see that the maximum latency is significantly increased, which shows that RTRG-RPC scheme does not violate priority scheduling and is still predicable in a lower priority.

TABLE I
RTRG-RPC Latency in Different Priorities (unit: cycles)

	Max	Min	Average	MSE
Highest Priority	8987	2037	2079.3	176.1
Lower Priority	179463	2036	2114.8	1267.2

B. Latency Comparison

In order to show the efficiency and the predictability of RTRG-RPC, we implement another two RPC policies in TZDKS. The first one is the traditional RPC method without any real-time consideration, and is labeled as TRG-RPC. TRG-RPC depends on the service process in GPOS, which runs under the standard idle-scheduling policy. The second policy is named ITRG-RPC, and is the enhanced version of TRG-RPC, which uses the interrupt handler as the RPC service in GPOS. In fact, we combine the event path model and GPOS service model of RTRG-RPC into a bundled implementation, called $Rapid\ Service$. Therefore, the ITRG-RPC is the enhanced version of TRG-RPC with the Rapid Service in GPOS. We measured the response time of RG-RPC under such three policies to evaluate the real-time performance of RTRG-RPC.

TABLE II
Latency Comparison for Three RG-RPCs (unit: cycles)

	Max	Min	Average	MSE
RTRG-RPC	6867	2043	2078.4.3	95.8
ITRG-RPC	372330	2298	64485.4	108405
TRG-RPC	50497874	49798712	49911274	125573.5

The results under these policies are listed in Table II. It can be learned that RTRG-RPC owns much higher efficiency and better predictability comparing to TRG-RPC and ITRG-RPC. We can also see that $Rapid\ Service$ in GPOS do more significant help to the efficiency than $Priority - Swapping$ because ITRG-RPC owns comparative minimum latency than that of RTRG-RPC. We note that the link in the GPOS is the most crucial link for the time efficiency of RTRG-RPC.

VI. Future Work

While this paper verifies the feasibility of obtaining a real-time service from a GPOS, a number of issues of details (such as cache miss and lock waiting, etc.) within the GPOS have been ignored. Our next plan is to address these issues and to extend the system model to a distributed multi-core platform.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (61602492, 61872444, 61502510). Much of the work reported in this paper took place while the first author was visiting the University of York.

References

[1] R. Berger. Consolidation in vehicle electronic architectures. *Think: Aact, Jul*, 2015.

[2] P. Dong, A. Burns, and Z. e. a. Jiang. Tzdks: A new trustzone-based dual-criticality system with balanced performance. In *RTCSA2018*, pages 59–64. IEEE, 2018.

[3] C. Hao, P. Cuifen, S. Jianhua, and S. Lin. Xenrpc: Design and implementation of security vm remote procedure call. *Journal of Computer Research and Development*, 5, 2012.

[4] Z. Jiang. *Real-Time I/O System for Many-core Embedded Systems*. PhD thesis, University of York, 2018.

[5] Z. Jiang and N. Audsley. Vcdc: The virtualized complicated device controller. In *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[6] Z. Jiang, N. Audsley, and P. Dong. Blueio: A scalable real-time hardware i/o virtualization system for many-core embedded systems. *ACM Transactions on Embedded Computing Systems*, 18(3):19, 2019.

[7] Z. Jiang, N. C. Audsley, and P. Dong. Bluevisor: A scalable real-time hardware hypervisor for many-core embedded systems. In *Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2018.

[8] S. Pinto and P. et al. Ltzvisor: Trustzone is the key. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 76, 2017.

[9] P. J. Prisaznuk. Arinc 653 role in integrated modular avionics (ima). In *27th Digital Avionics Systems Conference*. IEEE, 2008.

[10] D. Sangorrín, S. Honda, and H. Takada. Reliable and efficient dual-os communications for real-time embedded virtualization. *Information and Media Technologies*, 8(1):1–17, 2013.

[11] M. e. a. Sato. A hybrid operating system for a computing node with multi-core and many-core processors. *International Journal of Advanced Computer Science (IJACSci)*, 3(7), 2013.

[12] D. Shen. Exploiting trustzone on android. *Black Hat USA*, 2015.

[13] Q. Zhang and L. Liu. Shared memory optimization in virtualized cloud. In *8th International Conference on Cloud Computing*. IEEE, 2015.